А.О. Сухов

Пермский государственный университет
sukhov_psu@mail.ru

# СРЕДА РАЗРАБОТКИ ВИЗУАЛЬНЫХ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ МОДЕЛИРОВАНИЯ

С языками предметной области мы встречаемся достаточно часто. Макросы, shell-скрипты, языки построения пользовательского интерфейса, программы создания презентаций – это все примеры предметно-ориентированных языков, поскольку они создавались для некоторой конкретной предметной области.

DSL достаточно просты в применении и понятны пользователям, поскольку они оперируют терминами предметной области. Именно поэтому в настоящее время разработано большое число DSL, предназначенных для решения тех или иных задач. Применение таких языков значительно упрощает написание программ.

Однако при всех достоинствах DSL у них есть один большой недостаток – сложность разработки. Если языки общего назначения дают возможность программировать в любой предметной области, то в случае DSL это невозможно. Для каждой конкретной предметной области, а в некоторых случаях, даже для каждой задачи приходится создавать новый DSL.

Если предметная область достаточно проста и язык несложен, то создать его будет нетрудно. Более сложная предметная область и язык потребуют больших усилий, несмотря на то, что в настоящее время существуют и генераторы лексических анализаторов, и другие инструменты для создания компиляторов, которые облегчают работу программиста.

Еще одним недостатком предметно-ориентированных языков является то, что при их разработке приходится создавать собственные полнофункциональные среды для работы с ними, т.е. удобный графический редактор или редактор кода с возможностью подсветки синтаксиса языка, автоматического дополнения слов, выделения ошибок и др.

Для решения этих проблем в последнее время стал активно развиваться новый тип программного обеспечения – языковые инструментарии.

## Языковые инструментарии

*Языковой инструментарий* (language workbench) – это вид программного обеспечения, разработанный для облегчения создания языков предметной области. Использование при создании DSL языкового инструментария значительно упрощает процесс разработки языка.

Удобство использования языковых инструментариев проявляется и при внесении изменений в уже созданные DSL. Разработчику не придется разбираться в большом объеме кода парсера языка, а будет достаточно изменить описание DSL в языковом инструментарии.

Сегодня большое развитие получили визуальные DSL, поскольку диаграммы обладают большей наглядностью и понятностью не только для программистов, но и для экспертов в предметной области и пользователей, которые могут принимать участие в построении системы.

Существует большое число средств для разработки графических редакторов с возможностью задать собственную графическую нотацию, создать в автоматизированном режиме сам графический редактор с репозиторием, браузером модели и т.п. Такими средствами являются MetaEdit+ [4, 5], Microsoft DSL Tools [2], Eclipse Graphical Modeling Framework (GMF) [3].

Анализ средств создания DSL показал, что каждая из указанных выше технологий обладает как достоинствами, так и недостатками.

Так, благодаря использованию подхода, основанного на интерпретации метамоделей, а не на генерации на их основе исходного кода системы, технология MetaEdit+, в отличие от других технологий, позволяет вносить изменения в описание DSL во время работы системы.

К недостаткам MetaEdit+ можно отнести то, что эта DSM-платформа для экспорта моделей использует свой собственный формат файлов (MXT), который отличается от общепринятого стандарта – XML. А это сказывается на открытости данной технологии.

GMF является самым мощным языковым инструментарием из рассмотренных выше. Фактически все, что умеют выполнять другие платформы, умеет и она. Однако ее использование затруднено отсутствием документации, громоздкостью и сложностью, а также частым выходом новых версий. Фактически, GMF находится еще в стадии интенсивного развития.

Нередки случаи, когда DSL становятся частью других приложений. Поэтому еще одной важной характеристикой языковых инструментариев является отчуждаемость созданных с их помощью DSL от среды разработки.

Нужно отметить, что технологии DSL Tools, GMF сильно связаны с платформами разработки – MS Visual Studio, Eclipse соответственно.

Особенно технология DSL Tools, которая требуют для работы не только многочисленных библиотек времени исполнения, подобно GMF, но и среды разработки, поскольку созданный язык может быть использован лишь в MS Visual Studio.

Подводя итог, можно говорить о том, что предметно-ориентированные языки, созданные с помощью какой-либо из рассмотренных технологий не могут быть использованы в сторонних приложениях. В случае DSL Tools, GMF это происходит из-за необходимости наличия среды программирования, а в случае MetaEdit+ – из-за недостаточной открытости этой DSM-платформы.

Кроме того, ни одна из технологий, за исключением MetaEdit+, не позволяет создавать динамически настраиваемые языки. Поэтому для настройки языка необходимо сначала внести изменения в исходный код проекта создания DSL, перекомпилировать этот проект, а лишь потом с помощью нового языка изменить саму ИС.

Устранение рассмотренных недостатков при создании визуальных предметно-ориентированных динамически настраиваемых языков моделирования стало поводом для создания системы MetaLanguage, базовые элементы метаязыка которой рассмотрены в [1].

## Инструментальная среда разработки

Для работы с объектами метаязыка спроектирована среда разработки, которая включает следующие компоненты (рис. 1):

- графический редактор;
- браузер объектов;
- репозитарий;
- валидатор;
- генератор.

*Среда разработки* – это реализация общих сервисных функций создаваемой системы. Она обеспечивает интеграцию всех компонентов в единое целое. Среда включает в себя главное меню и панель инструментов с функциями создания, изменения, удаления объектов и всего проекта, настройки графических параметров: цветов, шрифтов, толщины линий и пр.

Рассмотрим описание каждого из компонентов инструментальной среды подробнее.

### *Графический редактор*

*Графический редактор* – представляет собой рабочую область для изображения диаграмм.

Назначением редактора является создание, изменение, удаление моделей, а также установление взаимосвязей между различными моделями.

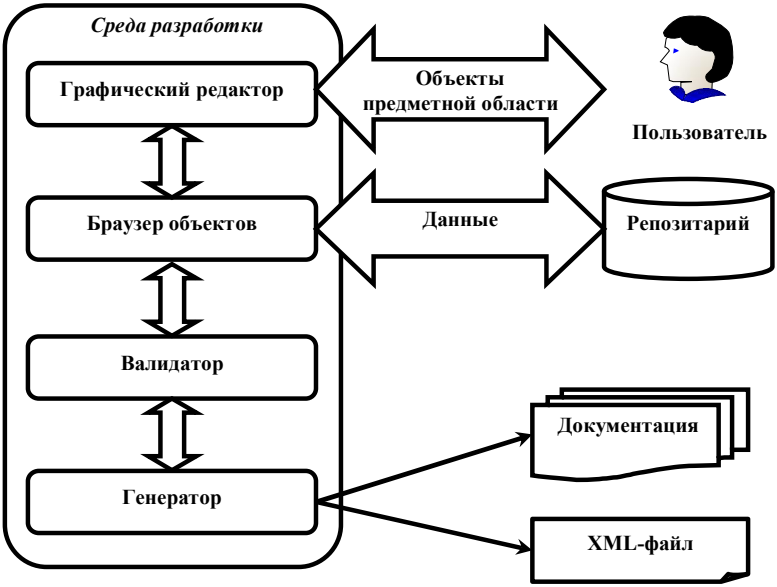


***Рис. 1. Архитектура системы MetaLanguage***

Каждая сущность модели представлена некоторым графическим символом, а отношения между сущностями изображаются различными видами линий.

Графический редактор предоставляет пользователю возможность располагать на рабочем листе различные фигуры (экземпляры сущностей и отношений), применять к этим фигурам наборы операций, задавать для них различные графические свойства.

## *Браузер объектов*

*Браузер* – инструментальное средство, предназначенное для просмотра и редактирования информации, хранящейся в репозитарии.

Браузер предоставляет возможность экспорта/импорта моделей из других систем. В качестве формата для экспорта/импорта моделей используется общепризнанный стандарт передачи структурированных данных – XML, содержащий помимо самих данных еще и метаданные,

которые описывают структуру хранящейся информации. В настоящее время большинство систем поддерживает возможность интерпретации XML-файлов.

MetaLanguage предоставляет в распоряжение пользователя четыре браузера:

- *браузер метамодели*, который позволяет просматривать и редактировать объекты метамодели;
- *браузер модели*, оперирующий объектами модели;
- *браузер объектов* позволяет пользователю работать с атрибутами сущностей и отношений модели.
- *браузер типов* позволяет просматривать существующие типы данных, создавать новые, выполнять операции над доменами допустимых значений.

### *Репозитарий*

Единым хранилищем всей информации о системе является *репозитарий*. Он содержит информацию о метамоделях, моделях, сущностях, отношениях, атрибутах, ограничениях, пиктограммах, используемых для отображения сущностей и отношений.

Репозитарий хранит информацию как о моделях, так и о метамоделях единообразно, что позволяет работать с ними с помощью одного инструментария.

Физически репозитарий представляет собой реляционную БД.

Репозитарий может работать в многопользовательском режиме. При этом для согласования действий пользователя при изменении БД используется механизм транзакций. То есть, если один из пользователей в определенный момент времени изменяет какую-либо информацию в БД, то доступ к этой информации для других пользователей закрыт. Как только первый пользователь подтвердит изменения, остальные пользователи смогут получить доступ уже к обновленной информации. Таким образом достигается актуальность и согласованность информации, хранимой в БД.

Кроме того, механизм транзакций используется для отмены всех изменений произведенных над моделью. При начале работы с моделью, для каждого пользователя создается сессия, с которой связана отдельная транзакция. Если пользователь после окончания работы не хочет сохранять изменения, то транзакция будет отменена, в результате чего никаких изменений в БД произведено не будет.

### *Валидатор*

Валидатор проверяет соответствие модели ограничениям, заданным пользователем. При выполнении проверки каждое ограничение будет применено к каждому экземпляру сущности и отношения. В случае если ограничения не выполняется, будет выдано сообщение об ошибке.

### *Генератор*

Генератор позволяют на основе имеющихся моделей сгенерировать XML-файл или документацию модели.

XML-файл будет содержать информацию о модели: свойства модели, сущности, отношения, их атрибуты, наложенные на модель ограничения.

Документация модели включает в себя: название модели, информацию о разработчиках, которые принимали участие в ее создании, графическое представление модели со ссылками на описание отдельных ее частей.

## Логика работы системы MetaLanguage

Рассмотрев основные компоненты системы, перейдем к описанию логики ее работы. Наибольший интерес представляют операции открытия, изменения и удаления метамодели, поскольку при этом необходимо производить аналогичные операции и над всеми моделями, созданными на основе этой метамодели. Кроме того, будет рассмотрена процедура проверки ограничений, накладываемых на сущности и отношения метамоделей.

### *Открытие существующей метамодели*

При открытии метамодели из репозитария будут загружены все модели, созданные с помощью этой метамодели, все сущности, отношения метамодели и соответствующих моделей. Алгоритм открытия метамодели представлен на рис. 2.

```
foreach (сущность in открываемая_метамодель)
{
    Загрузить_атрибуты(сущность);
    Загрузить_операции(сущность);
    Загрузить_ограничения(сущность);
}

foreach (отношение in открываемая_метамодель)
{
    Загрузить_атрибуты(отношение);
    Загрузить_ограничения(отношение);
}

foreach (модель in открываемая_метамодель)
{
    foreach (экземпляр_сущности in модель)
        Загрузить_значения_атрибутов(экземпляр_сущности);

    foreach (экземпляр_отношения in модель)
        Загрузить_значения_атрибутов(экземпляр_отношения);
}
```

***Рис. 2. Алгоритм открытия метамодели***

### *Удаление объектов метамодели*

При удалении сущности необходимо:

1. найти все экземпляры этой сущности во всех моделях, созданных с помощью метамодели, которой принадлежит удаляемая сущность;
2. для каждого из экземпляров сущности удалить все отношения, связанные с ним;
3. для каждой удаляемой сущности метамодели удалить из репозитария информацию обо всех отношениях, связанных с ней;
4. удалить сущность: удалить все ее атрибуты, операции, ограничения.

При удалении отношения необходимо:

1. найти все экземпляры отношения во всех моделях, созданных на основе изменяемой метамодели и удалить их;
2. удалить текущее отношение: удалить все его атрибуты и ограничения.

При удалении метамодели необходимо удалить все модели, созданные на ее основе, при этом удаление сущностей и отношений выполняется описанными выше способами.

### *Изменение метамодели*

При внесении изменений в метамодель для поддержания системы в согласованном состоянии необходимо внести изменения и в зависимые от нее модели. При изменении сущности (отношения) необходимо во всех моделях, созданных с помощью изменяемой метамодели, найти все экземпляры сущностей (отношений) данного типа и изменить их. Таким образом, алгоритм изменения сущности метамодели на псевдокоде может быть записан в виде:

```
if (изменяемая_сущность.модель.созданные_модели != NULL)
  foreach (модель in изменяемая_сущность.модель.созданные_модели)
      foreach (экземпляр_сущности in модель)
          if (экземпляр_сущности.тип == изменяемая_сущность)
              Изменить_сущность(экземпляр_сущности);

Изменить_сущность(изменяемая_сущность);
```

***Рис. 3. Алгоритм изменения модели***

### *Проверка ограничений*

Некоторые виды ограничений, описанных в [1] проверяется при создании сущностей и отношений, другие – непосредственно после создания модели. Рассмотрим проверку каждого из видов ограничений более подробно.

Проверка ограничений на значения атрибутов экземпляров сущностей происходит по окончании создания модели. При этом ограничения, наложенные на каждую сущность метамодели, будут проверены для всех их экземпляров. Алгоритм проверки ограничений представлен на рис. 4.

```
foreach (сущность in модель.метамодель)
    foreach (ограничение in сущность)
        foreach (экземпляр_сущности in сущ-
        ность.экземпляры_сущности(модель))
            if (Ограничение_не_выполняется(ограничение, экзем-
            пляр_сущности))
                Выдать_сообщение_об_ошибке();
```

***Рис. 4. Алгоритм проверки ограничений на значения атрибутов экземпляров сущности***

Уникальность имени сущности проверяется непосредственно при ее создании. При этом в случае, если задано ограничение «уникально в рамках метамодели», то будут просмотрены все экземпляры сущности во всех моделях, построенных на основе данной метамодели. Если значением свойства является «уникально в рамках модели», то достаточно просмотреть все экземпляры сущности текущей модели. Таким образом, алгоритм проверки уникальности имени экземпляра сущности на псевдокоде можно представить в виде:

```
switch (новая_сущность.уникальность)
{
    case "уникально в рамках метамодели":
        foreach (модель in новая_сущность.модель.метамодель)
            foreach (экземпляр_сущности in модель)
                if (экземпляр_сущности.имя == новая_сущность.имя)
                {
                    Выдать_сообщение_об_ошибке();
                    break;
                }
        Создать_сущность(новая_сущность);
        break;

    case "уникально в рамках модели":
        foreach (экземпляр_сущности in новая_сущность.модель)
            if (экземпляр_сущности.имя == новая_сущность.имя)
            {
                Выдать_сообщение_об_ошибке();
                break;
            }
        Создать_сущность(новая_сущность);
        break;

    case "неуникально":
        Создать_Сущность(новая_сущность);
        break;
}
```

***Рис. 5. Алгоритм проверки уникальности имени сущности***

Ограничения на количество экземпляров сущности в модели, как и ограничения на уникальность имени экземпляра проверяются непосредственно при создании экземпляра сущности. В случае если число уже созданных экземпляров сущностей определенного типа равно числу, указанному при создании сущности, то новый экземпляр создан не будет.

Ограничения на уникальность имени экземпляра отношения, проверяются непосредственно при их создании. Алгоритм проверки ограничений этого вида аналогичен алгоритму проверки ограничений на уникальность имени экземпляра сущности, приведенного на рис. 5, с той лишь разницей, что вместо обхода по экземплярам сущностей производится обход по экземплярам отношений.

Ограничения на типы соединяемых сущностей проверяется при создании экземпляров отношений. Как только пользователь проводит связь между двумя экземплярами сущностей, система определяет тип соединяемых сущностей и проверяет, существует ли в метамодели отношение между сущностями этого типа. Если отношение существует, то система позволит создать связь, в противном случае будет выдано сообщение об ошибке.

Проверка ограничений на значения атрибутов соединяемых экземпляров сущностей проводится по окончании создания модели. Алгоритм проверки ограничений данного типа представлен на рис. 6.

```
foreach (экземпляр_отношения in модель)
    foreach (ограничение in экземпляр_отношения)
    {
        if (Ограничение_не_выполняется(ограничение, экзем-
        пляр_отношения))
            Выдать_сообщение_об_ошибке();
    }
```

***Рис. 6. Алгоритм проверки ограничений, накладываемых на значения атрибутов соединяемых экземпляров сущностей***

## Заключение

В статье приведено описание инструментальной среды разработки визуальных языков моделирования, а также рассмотрены базовые операции работы с метамоделями и моделями.

Система MetaLanguage позволяет строить модели, которые достаточно подробно и точно описывают предметную область. Настолько подробно и точно, насколько это нужно в каждом конкретном случае. При этом для различных уровней детализации описаний используются одни и те же конструкции.

Среда разработки проста в использовании, поэтому с ней могут работать не только профессиональные программисты, но и бизнес-аналитики.

Для работы, как с моделями, так и с метамоделями используется один и тот же инструментарий, поэтому процесс создания модели может быть итеративным: создав некоторый язык, можно использовать

его как метаязык для создания другого языка, который в свою очередь также может быть использован как метаязык и т.п. Таким образом, разработчик получает в распоряжение достаточно мощный инструментарий для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования.

## Библиографический список


1. Лядова Л.Н., Сухов А.О. Языковой инструментарий системы MetaLanguage// Математика программных систем: Межвуз. сб. научн. тр. / Перм. ун-т. Пермь, 2008. С. 114-125.
2. *Anderson C.* DSL Tools: [Электронный документ] (http://www.codeproject.com/KB/cs/DSLTools.aspx). Проверено 21.05.2008.
3. Graphical Modeling Framework: [Электронный документ] (http://www.eclipse.org/gmf). Проверено 21.05.2008.
4. *Mazanek S.* Visual Languages. MetaEdit+: [Электронный документ] (http://visual-languages.blogspot.com/2007/11/metaedit.html). Проверено 21.05.2008.
5. *Tolvanen Juha-Pekka, Rossi Matti* MetaEdit+: defining and using domain-specific modeling languages and code generators: [Электронный документ] (http://portal.acm.org/citation.cfm?id=949365). Проверено 21.05.2008.